CENTRE DE RECHERCHE SUR LES TRANSPORTS
(C.R.T.)
CENTRE FOR RESEARCH ON TRANSPORTATION

UNIVERSITÉ DE MONTRÉAL
ÉCOLE DES HAUTES ÉTUDES COMMERCIALES
ÉCOLE POLYTECHNIQUE

CENTRE DE RECHERCHE SUR LES TRANSPORTS
UNIVERSITÉ DE MONTRÉAL
C.P. 6128, SUCCURSALE A,
MONTRÉAL
CANADA H3C 3J7

TÉLÉPHONE: (514) 343-7575
TÉLEX: (21) 05-561325
EMAIL: crt@crt.umontreal.ca
TÉLÉCOPIEUR (FAX): (514) 343-7121

91-02551/

CRT-741

The Delivery Man Problem and Cumulative Matroids

by
Matteo Fischetti[1]
Gilbert Laporte[2]
and
Silvano Martello[1]

[1] DEIS, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy.

[2] Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale A, Montréal, Canada H3C 3J7.

Centre de recherche sur les transports — Publication #741

January 1991

Dépôt légal – 1er trimestre 1991

Abstract

Given a completed directed graph *G=(V,A)*, the *Delivery Man Problem* (DMP) consists of determining a Hamiltonian circuit minimizing the sum of distances from a given vertex v_1, to every vertex of *V*, including v_1 itself. There exists a number of applications of the DMP in the fields of distribution and of machine scheduling. The DMP is NP-hard. The objective of this paper is to develop new theoretical results and an exact algorithm for the problem. New integer linear programming formulations are provided, and results on the matroidal structure of a class of combinatorial problems are developed. These are used to derive lower bounds for the DMP. These bounds are embedded into an enumerative algorithm. Computational results indicate that the proposed algorithm can solve to optimality problems involving up to 60 vertices. This compares favourably with previously published methods.

Keywords: Delivery Man Problem, Cumulative Matroids, Traveling Salesman Problem, Greedy Algorithm.

Résumé

Le *problème du livreur* (PL) consiste à déterminer un circuit hamiltonien sur un graphe *G=(V,A)*, de façon à minimiser la somme des distances d'un sommet donné v_1 à tout sommet de *V*, incluant v_1 lui-même. Ce problème possède plusieurs applications en distribution et en planification de la production. Le PL est un problème NP-dur. L'objectif de cet article est de développer de nouveaux résultats théoriques, ainsi qu'un algorithme exact pour ce problème. On propose de nouvelles formulations de programmation linéaire en nombres entiers, et l'on étudie la structure matroïdale d'une classe de problèmes d'optimisation combinatoire. Ces résultats servent à l'établissement de bornes inférieures pour le PL. Ces bornes sont par la suite imbriquées dans un algorithme énumératif. Les résultats numériques indiquent qu'on peut résoudre de façon exacte des problèmes comportant jusqu'à 60 sommets. Ces résultats se comparent avantageusement aux meilleurs résultats connus.

Mots-clés: problème du livreur, matroïdes cumulatives, problème du voyageur de commerce, algorithme glouton.

In this paper, we consider the ***Delivery Man Problem*** (DMP), a variant of the *Traveling Salesman Problem* (TSP). The DMP can be defined as follows. Let *G=(V,A)* be a complete directed graph with vertex set $V = \{v_1, ..., v_n\}$ and arc set $A = \{(v_1, v_j) : v_i, v_j \epsilon V,\ v_i \neq v_j\}$, having a positive distance or cost c_{ij} associated with each arc (v_i, v_j). The DMP consists of determining a Hamiltonian circuit so that the sum of all distances along the circuit from a given vertex v_1 to all vertices (including v_1) is minimized. More formally, we seek a permutation $C = (v_{i_1} = v_1, v_{i_2}, ..., v_{i_n})$ of V minimizing

$$z = \sum_{s=1}^{n} \sum_{k=1}^{s} c_{i_k i_{k+1}}, \quad \text{where } v_{i_{n+1}} = v_1.$$

Equivalently, the objective is to minimize

$$z = \sum_{k=1}^{n} (n - k + 1)\, c_{i_k i_{k+1}}. \tag{1}$$

Applications of the DMP include a number of delivery problems in which one seeks to minimize the average arrival time at the various customer locations. Pizza delivery is a common example. Another application is the routing of automated guided vehicles through n cells in a flexible manufacturing system (Simchi-Levi and Berman, 1991). A well-known machine scheduling problem treated by Rinnooy Kan (1976), and Picard and Queyranne (1978) can also be viewed as a DMP: the problem is to minimize mean flow-time for n jobs with change-over times, arriving at time zero, and to be processed sequentially on a single machine. In addition, a number of variants of the DMP have been described by Tsitsiklis (1990).

The complexity of the DMP has been addressed by Rinnooy Kan (1976), Sahni and Gonzalez (1976) and Afrati, Cosmadakis, Papadimitriou, Papageorgiou and Papakonstantinou (1986). The DMP is NP-hard in the strong sense, as can easily be seen by transformation of its recognition version from the *Hamiltonian Circuit Problem* (HC), known to be strongly NP-complete (Garey and Johnson, 1979). Indeed, given any instance of HC, relative to a graph $G^h = (V^h, A^h)$ with vertex set $V^h = \{v_i, ..., v_n\}$ and arc set A^h, define an instance of the DMP having $V = V^h$, $c_{ij} = 0$ if $(v_i, v_j) \epsilon A^h$, $c_{ij} = 1$ otherwise. Then G^h possesses a Hamiltonian circuit if and only if the solution value of this DMP instance is equal to zero. When the DMP is defined on a tree network, it can be solved in $O(n^p)$ time, where p is the number of pendant vertices (Minieka, 1989).

Simchi-Levi and Berman (1991) have proposed a branch-and-bound scheme for the DMP, based on a shortest spanning tree relaxation, and permitting the exact solution of problems with up to 20 vertices. A different enumerative algorithm has been provided by Lucena (1990). It is based on a compact non-linear integer programming formulation first proposed by Picard and Queyranne (1978) for the *Time-Dependent TSP*. Lucena computes lower bounds, through dynamic programming, by applying an ascent procedure to a Lagrangean relaxation of the problem. The largest problem solved exactly by this method contains 30 vertices. A similar approach has been independently proposed by Bianco, Mingozzi and Ricciardelli (1988), who report computational results for problems containing up to 35 vertices.

Our objective is to develop new theoretical results and an exact algorithm for the

problem. The remainder of this paper is organized as follows. New integer linear programming formulations are presented in Section 1. In Section 2, we develop results on the matroidal structure of a related class of combinatorial problems. These are used in Section 3 to derive lower bounds for the DMP, from relaxations of the models of Section 1. In Section 4, we describe an enumerative algorithm for the exact solution of the problem, based on the lower bounds of Section 3, a dominance criterion, and a heuristic algorithm. Computational results are presented in Section 5, and the conclusion follows in Section 6.

1. Integer Linear Programming Formulations

In addition to already known DMP formulations, the following can be used to derive lower bounds for the problem.

1.1 Path Formulations

A two-index formulation for the DMP is obtained by defining binary "arc variables" y_{ij} equal to 1 if and only if arc (v_i, v_j) is used in the optimal tour, and "path variables" t_i, equal to the length of the path from v_1 to v_i $(i = 2, ..., n)$ in the optimal solution; further, let t_1 be the length of the optimal Hamiltonian circuit. Assuming $y_{ij} = 0$ when $i = j$, the formulation is

$$\text{(DMP1)} \qquad \text{minimize} \quad \sum_{i=1}^{n} t_i \tag{2}$$

subject to

$$\sum_{j=1}^{n} y_{ij} = 1 \qquad (i = 1, ..., n) \tag{3}$$

$$\sum_{i=1}^{n} y_{ij} = 1 \qquad (j = 1, ..., n) \tag{4}$$

$$t_j \geq c_{1j} - M(1 - y_{1j}) \qquad (j = 2, ..., n) \tag{5}$$

$$t_j \geq t_i + c_{ij} - M(1 - y_{ij}) \qquad (i = 2, ..., n;\ j = 1, ..., n) \tag{6}$$

$$y_{ij} \epsilon \{0, 1\} \qquad (i, j = 1, ..., n), \tag{7}$$

where M is an arbitrarily large number.

Constraints (3) and (4) are standard degree constraints: they ensure that every vertex is left and entered exactly once. Constraints (5) impose conditions on the path lengths from v_1 to every other vertex. Constraints (6) stipulate that if v_1 immediately follows v_i in the tour, then $t_j \geq t_i + c_{ij}$; otherwise, they are ineffective. These constraints also act as subtour elimination constraints: following Miller, Tucker and Zemlin (1960), summing up these constraints over all arcs of a subtour not including v_1 yields a contradiction. Constraints (5) and (6), coupled with the objective function, ensure that in the optimal tour $(v_{i_1} = v_1, v_{i_2}, ..., v_{i_n})$, we have $t_{i_s} = t_{i_{s-1}} + c_{i_{s-1}, i_s}$ for $s = 2, ..., n$, and that $t_1 = t_{i_n} + c_{i_n, 1}$. As in Desrochers and Laporte (1991), applying a lifting process to (5) and (6) yields the stronger constraints

$$t_j \geq c_{1j} - M(1 - y_{1j}) + (M - c_{1j} - c_{j1})\, y_{j1} \qquad (i = 2, ..., n) \tag{5'}$$

$$t_j \geq t_i + c_{ij} - M(1 - y_{ij}) + (M - c_{ij} - c_{ji})\, y_{ji} \qquad (i = 2, ..., n;\ j = 1, ..., n). \quad (6')$$

One can readily check that in (6'), $y_{ij} = 1$ yields $t_j \geq t_i + c_{ij}$, $y_{ji} = 1$ yields $t_i \leq t_j + c_{ji}$, and that (6') is ineffective whenever $y_{ij} = y_{ji} = 0$. The same considerations apply to (5'). In no case is a feasible solution eliminated by these lifted constraints.

The model, as well as those of Section 1.2, can be strengthened further by introducing standard TSP subtour elimination constraints

$$\sum_{v_i, v_j \in S} y_{ij} \leq |S| - 1 \qquad (S \subset V;\ 2 \leq |S| \leq n - 2) \qquad (8)$$

or, for that matter, any valid inequality for the asymmetric TSP (see, e.g., Grötschel and Padberg, 1985).

1.2 Arc-Path Formulations

In the following formulation, variables x_{ij} are equal to 0 if (v_i, v_j) is not used, and to $n - k + 1$ if it appears in position k on the Hamiltonian tour. Binary variables y_{ij} are equal to 1 if and only if arc (v_i, v_j) appears on the circuit. We assume that $x_{ij} = y_{ij} = 0$ whenever $i = j$.

(DMP2) minimize $$\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\, x_{ij} \qquad (9)$$

subject to

$$\sum_{j=1}^{n} y_{ij} = 1 \qquad (i = 1, ..., n) \qquad (10)$$

$$\sum_{i=1}^{n} y_{ij} = 1 \qquad (j = 1, ..., n) \tag{11}$$

$$y_{ij} \epsilon \{0, 1\} \qquad (i, j = 1, ..., n) \tag{12}$$

$$\sum_{i=2}^{n} x_{i1} = 1 \tag{13}$$

$$\sum_{i=1}^{n} x_{ik} - \sum_{j=1}^{n} x_{kj} = \begin{cases} 1 - n & (k = 1) \\ 1 & (k = 2, ..., n) \end{cases} \tag{14}$$

$$x_{ij} \leq r_{ij}\, y_{ij} \qquad (i, j = 1, ..., n) \tag{15}$$

$$x_{ij} \geq 0 \quad \text{and integer} \quad (i, j = 1, ..., n), \tag{16}$$

where

$$r_{ij} = \begin{cases} 1 & if\ j = 1 \\ n & if\ i = 1 \\ n - 1 & otherwise \end{cases}$$

is an obvious upper bound on x_{ij}.

Constraints (10)-(12) are those of an assigment problem (AP), and the solution will therefore consist of a set of subtours. By constraints (15), x_{ij} can only take a positive value if v_j follows v_i on one of the subtours. On the other hand, constraints (14) ensure that no subtour disconnected from v_1 can exist. Indeed, suppose there was a subtour over a subset of $V \backslash \{v_1\}$: applying constraints (14) to all vertices v_k of the

subtour and summing them up would yield a contradiction. Therefore, the AP solution defines a Hamiltonian circuit visiting in sequence, say, vertices $v_{i_1} = v_1, v_{i_2}, \ldots, v_{i_n}$. By constraint (13), $x_{i_n,1}$ must be equal to 1 and, successively applying constraints (14) for $k = i_h$ $(h = 1, \ldots, n-1)$ yields $x_{i_h,i_{h+1}} = n - h + 1$.

Relaxations of this model can in some cases be strengthened by the introduction of surrogate constraints such as

$$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} = n(n+1)/2. \tag{17}$$

(DMP2) contains two well-known optimization problems: an assignment problem defined by constraints (10)-(12), and a network flow problem defined by constraints (13), (14) and (16). A possible lower bounding approach then consists of using Lagrangean decomposition (Ribeiro and Minoux, 1985; Jørnsten and Nasberg, 1986; Guignard and Kim, 1987), by relaxing coupling constraints (15) in a Lagrangean fashion.

A stronger model with the same decomposition property can be derived at the expense of an increase in the number of variables and constraints. Let binary variables x_{ij}^k take the value 1 if and only if arc (v_i, v_j) appears along the optimal tour on the path from v_i to v_k $(k = 2, \ldots, n)$. We assume $x_{ij}^k = 0$ if $i = j$, and also $x_{i1}^k = x_{ki}^k = 0$ for all k and i. Note that the x_{ij} variables of (DMP2) are equal to $y_{ij} + \sum_{k=2}^{n} x_{ij}^k$. The model is then

$$\text{(DMP3)} \quad \text{minimize} \quad \sum_{k=2}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\, x_{ij}^k + \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\, y_{ij} \tag{18}$$

subject to (10)-(12), and

$$\sum_{i=1}^{n} x_{ih}^{k} - \sum_{j=1}^{n} x_{hj}^{k} = \begin{cases} -1 & if\ h = 1 \\ 1 & if\ h = k \\ 0 & otherwise \end{cases} \qquad (h = 1, ..., n;\ k = 2, ..., n) \quad (19)$$

$$x_{ij}^{k} \leq y_{ij} \quad (i, j = 1, ..., n;\ k = 2, ..., n) \quad (20)$$

$$x_{ij}^{k} \in \{0, 1\} \quad (i, j = 1, ..., n;\ k = 2, ..., n). \quad (21)$$

Here, constraints (19) and (21) ensure that for every $k = 2, ..., n$, the x_{ij}^{k} variables define a path from v_1 to v_k, but this path is not necessarily simple. As in (DMP2), it is not difficult to see that the solution y consists of a single tour and, by constraints (20), the x_{ij}^{k} variables will take the value 1 or 0 according to whether arc (v_i, v_j) is on the path from v_1 to v_k or not.

Similar to (DMP2), this model lends itself to Lagrangean decomposition. Computational experiments with a branch-and-bound algorithm based on this technique were performed, and exact solutions were obtained for randomly generated DMP instances involving up to 30 vertices. In the following sections, we derive from these models better formulations yielding tighter lower bounds, permitting the exact solution of larger problem sizes.

2. Cumulative Combinatorial Optimization Problems and Matroids

In this section we introduce a generalization of the class of combinatorial optimization problems associated with a subset system, and show that the main result of matroid theory on the greedy algorithm applies to it. This will be used in Section 3 to obtain lower bounds for the DMP.

Recall that a *subset system* $S = (E, \mathcal{F})$ is a finite set E of items, together with a collection $\mathcal{F}$ of *feasible* (or *independent*) subsets of E, closed under inclusion (i.e., if $I_1 \subset I_2$ and $I_2 \epsilon \mathcal{F}$, then $I_1 \epsilon \mathcal{F}$). Without loss of generality, one can assume that $\{e\} \epsilon \mathcal{F}$ for all $e \epsilon E$. The subset system S is a *matroid* when for all pairs $I, J \epsilon \mathcal{F}$ with $|I| = |J| + 1$, there exists $e \epsilon I \backslash J$ such that $J \cup \{e\} \epsilon \mathcal{F}$ (see, e.g., Lawler, 1976).

The *cumulative combinatorial optimization problem* associated with S is defined as follows. Given the cost c_j of each item $j \epsilon E$, and $n (\leq |E|)$ real multipliers $a_1, ..., a_n$ satisfying $a_1 \geq a_2 \geq ... \geq a_n > 0$, determine a permutation $(x_j : j \epsilon E)$ of $(a_1, ..., a_n, \ 0, ..., 0)$ such that the subset of items with non-zero x_j values belongs to $\mathcal{F}$, and the *cumulative cost* $\sum_{j \epsilon E} c_j x_j$ is minimized. Note that adding a constant K to all c_j, $j \epsilon E$, changes the cumulative cost of all feasible solutions x by the same quantity $K \sum_{i=1}^{n} a_i$. Hence, we will assume without loss of generality that $c_j \geq 0$ for all $j \epsilon E$.

When $a_1 = ... = a_n = 1$, we have the *non-cumulative* (classical) combinatorial optimization problem (in minimization form) associated with S, calling for a minimum cost subset $I^* \epsilon \mathcal{F}$ with $|I^*| = n$. (The classical theory is usually developed for maximization problems. Observe that if we remove from $\mathcal{F}$ all its members I such that $|I| > n$, which play no role in the optimization, the non-cumulative problem

can equivalently be formulated as that of determining a subset $I^* \epsilon \mathcal{F}$ maximizing $\sum_{j \epsilon I^*} p_j$, where positive "profits" p_j, $j \epsilon E$, are defined as $M - c_j$, and M is a very large positive value.)

Solving a cumulative optimization problem involves two related decisions:

1) finding a feasible $I \epsilon \mathcal{F}$ such that $|I| = n$, say $I = \{e_1, ..., e_n\}$; and
2) determining a permutation $(\pi(k) : 1 \leq k \leq n)$ of $(1, ..., n)$ such that $\sum_{k=1}^{n} a_k c_{e_{\pi(k)}}$ is minimized.

Assuming the first decision has been taken, the second poses no difficulty. Indeed, an interchange argument shows that the quantity $\sum_{k=1}^{n} a_k c_{e_{\pi(k)}}$ is minimized when $c_{e_{\pi(1)}} \leq c_{e_{\pi(2)}} \leq \ ... \ \leq c_{e_{\pi(n)}}$ (see the proof of Theorem 1, later in this section). Note that π is independent of the a_k multipliers. On the other hand, the first decision heavily depends on these values. Indeed, define

$$\delta_k := a_k - a_{k+1} \geq 0$$

for $k = 1, ..., n$ (assuming $a_{n+1} := 0$). The minimum cumulative cost associated with I can then be computed as $\sum_{k=1}^{n} \delta_k \, w_k(I)$, where $w_k(I) := \sum_{i=1}^{k} c_{e_{\pi(i)}}$ is the sum of the k smallest costs associated with the items in I. (Note that $\sum_{k=1}^{n} \delta_k \, w_k(I) = \delta_1 \, c_{e_{\pi(1)}} + \delta_2 (c_{e_{\pi(1)}} + c_{e_{\pi(2)}}) + \ ... \ + \delta_n (c_{e_{\pi(1)}} + \ ... \ + c_{e_{\pi(n)}}) = c_{e_{\pi(1)}} (\delta_1 + \ ... \ + \delta_n) + c_{e_{\pi(2)}} (\delta_2 + \ ... \ + \delta_n) + \ ... \ + c_{e_{\pi(n)}} \delta_n = c_{e_{\pi(1)}} \, a_1 + \ ... \ + c_{e_{\pi(n)}} \, a_n$.) Hence, one is interested in choosing I so as to have all $w_k(I)$ as small as possible, whereas in a non-cumulative problem only $w_n(I)$ is considered. As an example, consider the subset system $S = (E, \mathcal{F})$ defined on the digraph *G=(V,A)* of Figure 1, where the arc

costs are also given. Here $E := A$ (items are the arcs of G), $n := 5$, and $\mathcal{F}$:={$I \subseteq A$: no two arcs in I have the same head nor the same tail}. In this case we have only 2 feasible sets of cardinality n, namely

$$I_1 = \{(1,2),(2,3),(3,1),(4,5),(5,4)\}$$

$$I_2 = \{(2,4),(1,2),(3,1),(4,5),(5,3)\},$$

whose associated cumulative costs are $z_1 = 10 \sum_{k=1}^{5} a_k$ and $z_2 = 3a_1 + 10 \sum_{k=2}^{4} a_k + 21a_5$, respectively. Hence, $z_1 - z_2 = 7\,a_1 - 11\,a_5$, i.e., the optimal set is I_1 when $a_1 \leq 11\,a_5/7$ (e.g., $a_1 = a_2 = a_3 = a_4 = a_5 = 1$), and I_2 when $a_1 \geq 11\,a_5/7$ (e.g., $a_1 = 2$, $a_2 = a_3 = a_4 = a_5 = 1$).

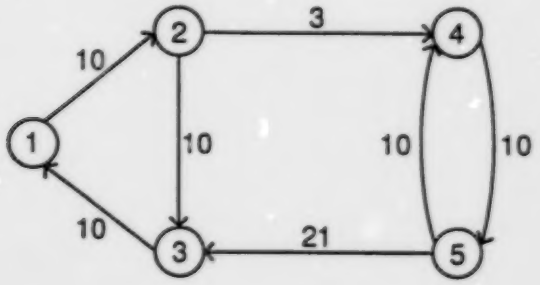

Figure 1. Example of a cumulative problem on a digraph

It is known that the greedy algorithm exactly solves any instance of the non-cumulative problem associated with a subset system S if and only if S is a matroid. A similar result holds for the cumulative case. Indeed, consider the following greedy algorithm for a cumulative problem:

procedure C_GREEDY:

begin

I:=$\emptyset$;

k:=0;

for each $j \epsilon E$ **by increasing** c_j **do**

if $k < n$ **and** $I \cup \{j\} \epsilon \mathcal{F}$ **then**

begin

I:=$I \cup \{j\}$;

$k := k + 1$;

$x_j := a_k$

end

else $x_j := 0$

end

and observe that when $a_1 = ... = a_n = 1$, this procedure coincides with the classical greedy algorithm.

Theorem 1. Algorithm C_GREEDY exactly solves the cumulative problem for all $(c_j : j \epsilon E)$, for all $n \leq |E|$ and for all multipliers $a_1 \geq a_2 \geq ... \geq a_n > 0$, if and only if the associated subset system is a matroid.

Proof. We first prove the "only if" part of the theorem. Assume that $S = (E, \mathcal{F})$ is not a matroid, i.e., there exist $I, J \epsilon \mathcal{F}$ with $|I| = |J| + 1$, such that no $e \epsilon I \backslash J$ satisfies $J \cup \{e\} \epsilon \mathcal{F}$. Hence consider the following input data for the cumulative optimization problem: $c_j := 0$ if $j \epsilon J$, $c_j := 1$ if $j \epsilon I \backslash J$, $c_j := \infty$ otherwise; $n := |I|$; and $a_1 \geq a_2 \geq ... \geq a_n > 0$ arbitrary finite values. The cumulative problem clearly has

a finite optimal value, since the cumulative cost associated with I is finite. On the other hand, procedure C_GREEDY will first set $x_j := a_k$ $(k = 1, ..., n-1)$ for $j \epsilon J$; then the items $e \epsilon I \backslash J$ will be discarded, and the procedure will either set $x_j := a_n$ for an item j with $c_j = \infty$, or even fail to determine a feasible set with cardinality n. In both cases, it does not provide the optimal solution.

Suppose now that $S = (E, \mathcal{F})$ is a matroid, and consider any possible input data of the associated cumulative problem; without loss of generality, we assume $c_i \neq c_j$ for all $i \neq j$. We prove that procedure C_GREEDY exactly solves this problem. Let $(x_j : j \epsilon E)$ be the solution computed by C_GREEDY, and $I := \{j \epsilon E : x_j > 0\} = \{e_1, ..., e_n\}$ the corresponding feasible set, where $c_{e_1} < c_{e_2} < \ ... \ < c_{e_n}$. The proof is by contradiction. Let $(y_j : j \epsilon E)$ be an optimal solution to the cumulative problem, $J := \{j \epsilon E : y_j > 0\} = \{b_1, ..., b_n\}$ the corresponding optimal set with $c_{b_1} < c_{b_2} < \ ... \ < c_{b_n}$, and suppose that $\sum_{j \epsilon E} c_j \, y_j < \sum_{j \epsilon E} c_j \, x_j$.

We first show that $\sum_{j \epsilon E} c_j \, y_j = \sum_{k=1}^{n} a_k \, c_{b_k}$, i.e., that $y_{b_k} = a_k$ for $k = 1, ..., n$. Suppose not, and let h be the smallest index such that $\beta := y_{b_h} \neq a_h$ (hence $\beta < \alpha_h$) and $q \epsilon \{h+1, ..., n\}$ be an index such that $y_{b_q} = a_h$. Then setting $y_{b_h} := a_h$ and $y_{b_q} := \beta$ would produce a feasible solution with cumulative cost changed by the amount $\left(a_h \, c_{b_h} + \beta \, c_{b_q}\right) - \left(\beta \, c_{b_h} + a_h \, c_{b_q}\right) = (\beta - a_h)(c_{b_q} - c_{b_h}) < 0$. This is impossible because of the optimality of y.

We are now in the position of proving that y cannot have a smaller cumulative cost of x. Indeed, it is known from matroid theory that $c_{e_k} \leq c_{b_k}$ holds for $k = 1, ..., n$

(see, e.g., Theorem 6.2 in Lawler, 1976). Hence

$$\sum_{j \epsilon E} c_j x_j = \sum_{k=1}^{n} a_k c_{e_k} \leq \sum_{k=1}^{n} a_k c_{b_k} = \sum_{j \epsilon E} c_j y_j,$$

a contradiction. □

Hence, when matroids are considered, a set *I* optimizing the non-cumulative cost is optimal for the cumulative problem as well.

Another well-known result for the non-cumulative matroid problem is that it can be formulated as a linear program (Edmonds, 1970, 1971). In the Appendix, we show that this result extends to the cumulative case.

3. Lower Bounds

We now derive lower bounds for the DMP by exploiting the structures analyzed in the previous section. Undirected and directed graphs will be considered separately.

3.1 Undirected graphs

Assume the cost matrix (c_{ij}) is symmetrical, so that the associated DMP can more appropriately be defined on an undirected graph *G=(V,E)*, where E is now a set of *edges* (v_i, v_j). E can be partitioned into $\{E_1, E_2\}$, where $E_1 = \{(v_i, v_j) \epsilon E : v_i = v_1 \text{ or } v_j = v_1\}$. The problem is then to minimize $\sum_{(v_i,v_j)\epsilon E_1} c_{ij} x_{ij} + \sum_{(v_i,v_j)\epsilon E_2} c_{ij} x_{ij}$, where the x_{ij}'s are defined as in Section 1.2, subject to

$$(x_{ij} : (v_i, v_j) \epsilon E_1) \text{ is a permutation of } (n, 1, 0, ..., 0), \tag{22}$$

$$(x_{ij} : (v_i, v_j)\epsilon E_2) \text{ is a permutation of } (n-1, n-2, ..., 2, 0, ..., 0), \qquad (23)$$

$$\{(v_i, v_j)\epsilon E_2 : x_{ij} > 0\} \text{ defines a tree spanning } V\backslash\{v_1\}, \qquad (24)$$

$$\sum_{v_i \epsilon V\backslash\{v_h, v_k\}} x_{ik} \geq x_{kh} - 1 \qquad (v_h, v_k \epsilon V;\ v_k \neq v_1). \qquad (25)$$

The validity of requirements (22), (23) and (24) poses no difficulty. Constraints (25) can be explained as follows. When $x_{kh} = 0$, the constraint is trivially satisfied. When, on the other hand, (v_k, v_h) belongs to the solution, then there must be another edge (v_i, v_k) with $x_{ik} = x_{kh} - 1$ or $x_{kh} + 1$. (Note that $v_k \neq v_1$ is assumed.) The formulation is also complete since the non-zero variables x_{ij} induce a spanning v_1-tree of G which contains no leaves v_k (with predecessor v_h, say) since otherwise the corresponding constraint (25) would be violated. Therefore, the v_1-tree is a Hamiltonian tour, visiting in sequence, say, vertices $v_1 = v_{i_1}, ..., v_{i_n}$, with $x_{i_1, i_2} = n$; hence when $k = i_j$ and $h = i_{j-1}$ $(j = 2, ..., n)$, constraint (25) implies $x_{i_j, i_{j+1}} = x_{i_{j-1}, i_j} - 1$.

If (25) is relaxed in a Lagrangean fashion, the problem separates into the problem induced by (22) (trivially solvable), and that induced by (23) and (24), defining the cumulative version of the well-known *graphic matroid* (see, e.g., Papadimitriou and Steiglitz, 1982). Therefore, the latter problem can be solved exactly by algorithm C_GREEDY of Section 2.

To summarize, the relaxed problem (22)-(24) can be solved through the following $O(n^2)$ algorithm: Find the two minimum-cost edges incident to v_1 and set $x_{ij} = n$ for the smallest one, and $x_{ij} = 1$ for the other; then determine a shortest spanning tree on the subgraph induced by $V\backslash\{v_1\}$, sort its $n-2$ edges, and define $x_{ij} = n-k$ for the k^{th} smallest such edge ($k = 1, ..., n-2$).

3.2 Directed graphs

For the directed case, a formulation of the DMP in the spirit of that of Section 3.1 is as follows. Let the arc set A be partitioned into $\{A_1, A_2\}$, with $A_1 = \{(v_i, v_j) \epsilon A : v_j = v_1\}$. The problem is then to minimize $\sum_{(v_i,v_j)\epsilon A_1} c_{ij}\, x_{ij} + \sum_{(v_i,v_j)\epsilon A_2} c_{ij}\, x_{ij}$, subject to

$$(x_{ij} : (v_i, v_j)\epsilon A_1) \text{ is a permutation of } (1, 0, ..., 0), \tag{26}$$

$$(x_{ij} : (v_i, v_j)\epsilon A_2) \text{ is a permutation of } (n, n-1, ..., 2, 0, ..., 0), \tag{27}$$

$$((v_i, v_j)\epsilon A_2 : x_{ij} > 0) \text{ contains exactly one arc entering each vertex } v_j \epsilon V\backslash\{v_1\}, \tag{28}$$

$$\sum_{v_i \epsilon V} x_{ik} - \sum_{v_j \epsilon V} x_{kj} = d_k := \begin{cases} 1-n & (k=1) \\ 1 & (k = 2, ..., n). \end{cases} \tag{29}$$

The validity of (26), (27) and (28) is straightforward, while constraints (29) are the same as constraints (14) introduced for model (DMP2). Again, the formulation is

complete since constraints (29) prevent the occurrence of leaves and of subtours disconnected from v_1. Moreover, using an argument similar to that of Section 3.1, it can be shown that the x_{ij} variables corresponding to the resulting Hamiltonian tour will take the proper values.

Let us introduce the following additional (redundant) constraints to the formulation:

$$\sum_{v_i \notin S} \sum_{v_j \in S} x_{ij} \geq \sum_{v_i \in V} x_{ik} \qquad (S \subseteq V \backslash \{v_1\}, v_k \in S). \tag{30}$$

In order to show these constraints are redundant, denote by p_i the position of vertex $v_i \in V \backslash \{v_1\}$ on the DMP tour and observe that the right-hand side of (30) is equal to $n - p_k + 1$: the first arc (v_t, v_h) of the optimal tour entering S must clearly satisfy $x_{th} = n - p_h + 1 \geq n - p_k + 1$, i.e. $x_{th} \geq \sum_{v_i \in V} x_{ik}$.

If we now relax constraints (29) and (30) in a Lagrangean fashion, the problem separates into two independent subproblems: the first is the trivial problem associated with constraints (26), and the second with constraints (27) and (28). The latter is the cumulative version of the *head partition matroid* in a directed graph (see, e.g., Papadimitriou and Steiglitz, 1982), and hence it is solved exactly by the greedy algorithm of Section 2.

An overall $O(n^2)$-time algorithm for the solution of relaxation (26)-(28) is then as follows. Find for every $v \in V$ the minimum-cost entering arc, and define $x_{ij} = 1$ for the arc entering v_1. Sort the remaining $n - 1$ arcs, and define $x_{ij} = n - k + 1$ for the k^{th} smallest such arc $(k = 1, ..., n - 1)$.

The Lagrangean objective function can be written as

$$\sum_{(v_i,v_j)\epsilon A} c_{ij}\, x_{ij} + \sum_{v_k \epsilon V} \alpha_k \left(d_k + \sum_{v_j \epsilon V} x_{kj} - \sum_{v_i \epsilon V} x_{ik} \right)$$
$$+ \sum_{S \subseteq V \setminus \{v_1\}} \sum_{v_k \epsilon S} \beta_k^S \left(\sum_{v_i \epsilon V} x_{ik} - \sum_{\substack{v_i \notin S \\ v_j \epsilon S}} x_{ij} \right)$$
$$= \sum_{v_k \epsilon V} \alpha_k\, d_k + \sum_{(v_i,v_j)\epsilon A} c'_{ij}\, x_{ij},$$

where α_k (unrestricted in sign) and $\beta_k^S \geq 0$ are the multipliers associated with constraints (29) and (30), respectively, and for each $(v_i, v_j)\epsilon A$

$$c'_{ij} = c_{ij} + \alpha_i - \alpha_j + \sum_{\substack{S \subseteq V \setminus \{v_1\}: \\ v_j \epsilon S}} \beta_j^S - \sum_{\substack{S \subseteq V \setminus \{v_1\}: \\ v_i \notin S, v_j \epsilon S}} \sum_{v_k \epsilon S} \beta_k^S.$$

The role of α_k is easily understood: the costs of the arcs entering v_k are decreased by α_k, whereas those of the arcs leaving v_k are increased by the same quantity. The effect of the β_k^S multipliers is to increase the costs of the arcs entering v_k and to decrease those of the arcs entering S. Thus the net result is an increase of the cost of all arcs entering v_k from inside S, at the expense of the arcs entering S but not v_k.

Since there are exponentially many β_k^S multipliers, we have developed a specialized heuristic approach for the solution of the Lagrangean dual problem, in which subgradient optimization is used to determine the α_k's, while a technique akin to dual ascent (see, e.g., Wong, 1984 and Balakrishnan, Magnanti and Wong, 1989) is used for the β_k^S multipliers. The approach can be summarized as follows:

Algorithm LB

1. initialize the α_k multipliers to convenient values (e.g. $\alpha_k := 0$ for all $v_k \epsilon V$);

   repeat

2. solve the Lagrangean relaxation, assuming $\beta_k^S = 0$ for all k and S, and let x^* be the optimal solution;

3. use a dual ascent technique to determine, on the basis of x^*, appropriate positive values of β_k^S for heuristically determined S and k (during this operation, x^* is updated so as to take into account the cost changes due to the Lagrangean variables β_k^S);

4. update $\alpha_k := \alpha_k + t\left(d_k + \sum_{v_j \epsilon V} x_{kj}^* - \sum_{v_i \epsilon V} x_{ik}^*\right)$ for all $v_k \epsilon V$, where t is an appropriate positive step size

   until termination condition is reached;

5. **return** the best lower bound found.

Steps 2 and 3 were implemented using the following procedure, which starts by implicitly determining the optimal solution x^* to the relaxed problem. This solution is then used (and implicitly updated) in the while loop, so as to increase its associated lower bound. The final x^* and its value δ are then computed at the end of the procedure.

procedure DUAL_ASCENT:

input: the original costs c_{ij} and the current Lagrangean multipliers α_k;

output: a lower bound value δ and the corresponding solution x^*;

begin

define the current Lagrangean costs $c'_{ij} := c_{ij} + \alpha_i - \alpha_j$;

comment: initialize A_0 to contain, for each $v_j \neq v_1$, the entering arc of minimum cost e_j;

for each $v_j \epsilon V \backslash \{v_1\}$ **do**

begin

$e_j := \min \{c'_{ij} : 1 \leq i \leq n,\ i \neq j\}$;

$A_0 := A_0 \cup \{(v_i, v_j) : c'_{ij} = e_j\}$

end;

while $G_0 = (V, A_0)$ does not contain a spanning arborescence rooted at v_1 **do**

begin

comment: determine a subset S yielding positive penalties $\beta_k^S (v_k \epsilon S)$;

let $S \subseteq V \backslash \{v_1\}$ be a maximal strongly connected component of G_0

such that no arc of A_0 enters S;

compute $u^S := \min \left\{ c'_{ij} - e_j : v_i \notin S \text{ and } v_j \epsilon S \right\}$;

apply procedure BETA (to be described later) to compute convenient

non-negative values β_k^S for all $v_k \epsilon S$, satisfying $\sum_{v_k \epsilon S} \beta_k^S = u^S$;

comment: update c'_{ik}, e_k $(v_k \epsilon S)$ and A_0;

for each $v_k \epsilon S$ **do**

begin

$e_k := e_k + \beta_k^S$;

for each $v_i \epsilon S \backslash \{v_k\}$ **do** $c'_{ik} := c'_{ik} + \beta_k^S$;

for each $v_i \notin S$ **do**

begin

$c'_{ik} := c'_{ik} + \beta_k^S - u^S$;

if $c'_{ik} = e_k$ **then** $A_0 := A_0 \cup \{(v_i, v_k)\}$

end

end

end;

comment: apply the greedy algorithm with respect to the final Lagrangean costs;

determine a permutation $(\pi(2), ..., \pi(n))$ of $(2, ..., n)$ such that $e_{\pi(2)} \leq e_{\pi(3)} \leq ... \leq e_{\pi(n)}$;

compute $e_1 := \min\{c'_{i1} : 2 \leq i \leq n\}$;

$$\delta := \left(\sum_{k=2}^{n} (n - k + 2)\, e_{\pi(k)} \right) + e_1 + \sum_{v_k \in V} \alpha_k\, d_k;$$

determine the solution x^* corresponding to δ (breaking ties by choosing the arcs of A_0 in such a way that they form a spanning arborescence rooted at v_1)

end.

Note that at any iteration, A_0 contains for each $v_k \neq v_1$, all minimum-cost arcs entering v_k, with respect to the current Lagrangean costs c'_{ij}, and that e_k is the corresponding minimum cost. Also observe that our particular choice of S is

motivated by the fact that when applied to this set, the positive multiplies $\beta_k^S (v_k \epsilon S)$ will increase the costs of the arcs of A_0 entering v_k, and hence the overall lower bound δ. On the other hand, choosing a set S with arcs $(v_i, v_k) \epsilon A_0$ and entering S does not guarantee a higher lower bound since the cost of these arcs is decreased by $u^S - \beta_k^S$. Finally, note that $u^S \left(= \sum_{v_k \epsilon S} \beta_k^S \right)$ is computed in such a way that all current arcs of A_0 remain least cost entering arcs.

One can easily see that the values of u^S computed by the above procedure (as well as arc set A_0) are independent of the β_k^S values computed by procedure BETA, since the same quantity β_k^S is added to the cost of all arcs entering v_k, and hence to e_k. In this respect, our procedure is similar to Edmond's algorithm (1967) for determining the minimum cost arborescences of G rooted at v_1 (see the $O(n^2)$ Fischetti and Toth implementation, 1989). In particular, the values u^S and the set A_0 computed in our procedure coincide with the dual variables u_k^* and the arc set A_0 in Fischetti and Toth. It follows that procedure DUAL_ASCENT can be implemented to run in $O(n^2)$ time.

Procedure BETA

We now describe procedure BETA for the computation of the β_k^S multipliers. This is conceived so as to "split" u^S among the β_k^S associated with the vertices v_k of S in such a way that the resulting values e_k are as close as possible. This choice produces a larger value of the overall lower bound δ. Indeed, from the definition of this bound, it can be seen that δ will be larger if the $e_{\pi(k)}$ values are as close as possible in size.

Assume that the vertices of S are $v_{k_1}, \ldots, v_{k_{|S|}}$, sorted so that $e_{k_1} \leq e_{k_2} \leq \ldots \leq e_{k_{|S|}}$, and introduce a dummy vertex $v_{k_{|S|+1}}$ with $e_{k_{|S|+1}} = \infty$. A possible

implementation of procedure BETA could consist of determining the β_k^S values so as to increase the value of e_{k_1} to that of e_{k_2}, then increase the value of both e_{k_1} and e_{k_2} to that of e_{k_3}, and so on, until $\sum_{v_k \epsilon S} \beta_k^S = u^S$. This corresponds to determining the (unique) index h such that

$$L(h) = \left(\sum_{i=1}^{h} e_{k_i} + u^S \right) / h \leq e_{h+1},$$

hence defining

$$\beta_{k_i}^S = L(h) - e_{k_i} \tag{31}$$

for $i = 1, ..., h$, and $\beta_{k_i}^S = 0$ for $i = h+1, ..., |S|$. Such a procedure would require $O(|S| \log |S|)$ time for the initial sorting, plus $O(|S|)$ time for the remaining computations. In this way, the overall time complexity of procedure DUAL_ASCENT would increase to $O(n^2 \log n)$. It is not so difficult, however, to obtain an overall $O(|S|)$ time complexity, by employing a median-finding technique similar to that used for knapsack problems (see, e.g., Martello and Toth, 1990, Section 2.2.2). The resulting procedure BETA will partition S into subsets S_1 and S_2 such that $L(h) = \left(\sum_{v_k \epsilon S_1} e_k + u^S \right) / |S_1|$, and $\beta_{k_i}^S$ is given by (31) if $v_{k_i} \epsilon S_1$, and takes the value 0 if $v_{k_i} \epsilon S_2$.

Example

In order to illustrate the procedure, consider the example depicted in Figure 2. All graphs show only those arcs relevant to our computations; all missing arcs have a cost larger than 10. We assume $\alpha_k = 0$ for all k. Arcs belonging to A_0 at the end of each iteration are shown in bold lines. The first graph shows the arcs selected in

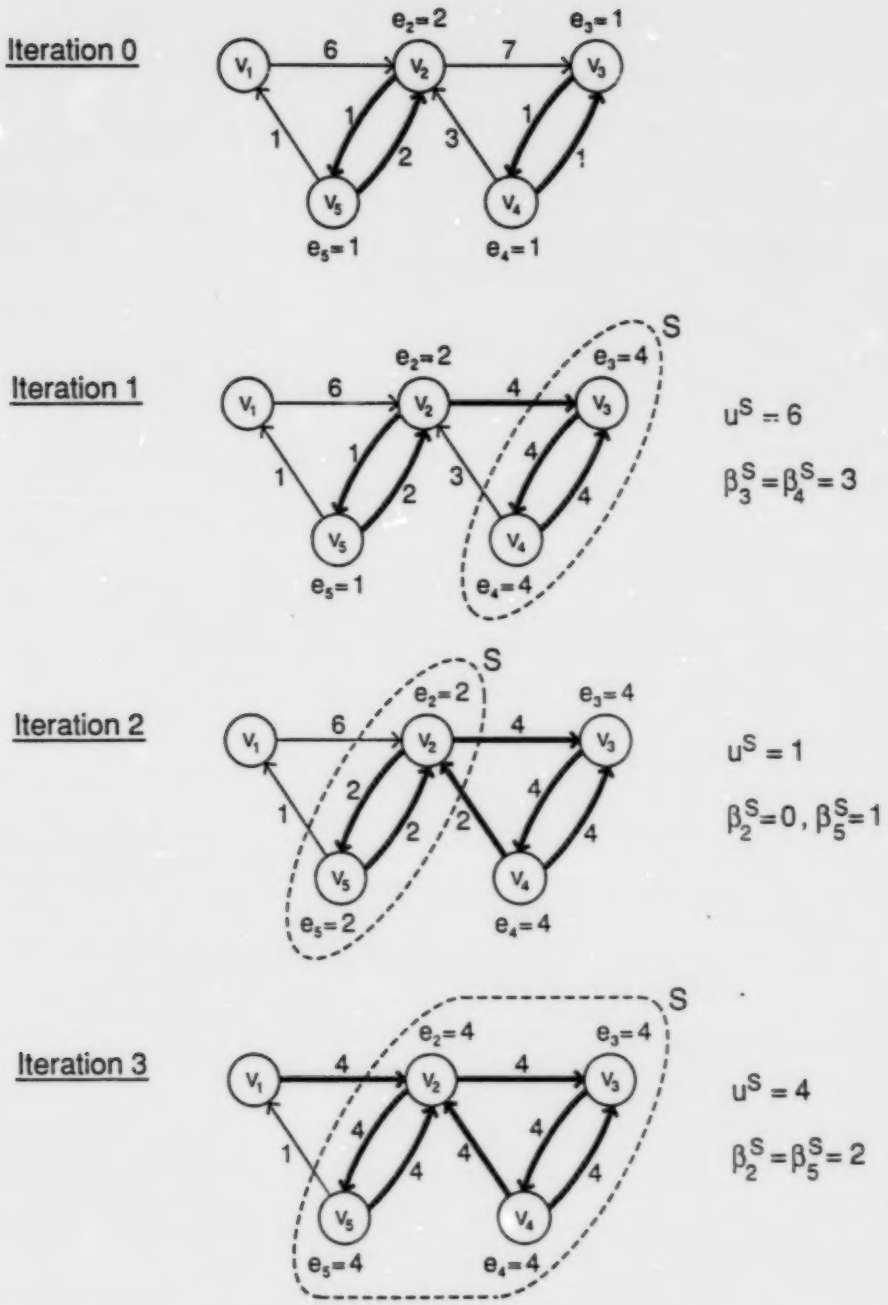

Figure 2. Illustration of the lower bound computation in procedure DUAL_ASCENT.

the initial computation of the e_j values. At iteration 1, S is defined as $\{v_3, v_4\}$, $u^S = 6$, $\beta_3^S = \beta_4^S = 3$, e_3 and e_4 become equal to 4 and arc (v_2, v_3) is inserted in A_0 with cost $c'_{2,3} = 7 + 3 - 6 = 4$. At iteration 2, $S = \{v_2, v_5\}$, $u^S = 1$, $\beta_2^S = 0$, $\beta_5^S = 1$, e_5 becomes equal to 2 and arc (v_4, v_2) enters A_0 with cost $c_{4,2} = 3 + 0 - 1 = 2$. Finally, at iteration 3, $S = \{v_2, v_3, v_4, v_5\}$, $u^S = 4$, $\beta_2^S = \beta_5^S = 2$, $\beta_3^S = \beta_4^S = 0$, e_2 and e_5 become equal to 4, and (v_1, v_2) enters A_0 with cost $c'_{1,2} = 6 + 2 - 4 = 4$. At this stage, the termination condition is satisfied, and the resulting lower bound is $\delta = 5e_2 + 4e_3 + 3e_2 + 2e_1 + 1 = 57$. A possible solution x^* associated with δ (to be used for the subgradient updating of Lagrangean variables α_k) is $x^*_{1,2} = 5$, $x^*_{2,3} = 4$, $x^*_{3,4} = 3$, $x^*_{2,5} = 2$, $x^*_{5,1} = 1$, and $x^*_{ij} = 0$ otherwise. Note that computing this lower bound using the initial c_{ij}'s yields a value of $5\times1 + 4\times1 + 3\times1 + 2\times2 + 1 = 17$. □

4. Enumerative Algorithm

The results of the previous sections were combined to obtain an exact algorithm for the DMP.

4.1 General Description

The strategy consists of iteratively extending a simple path emanating from vertex v_1 through a depth-first branch-and-bound search. As in Fischetti and Toth (1988), the following dominance criterion is first applied at every node of the branch-decision tree. Let $P = \{v_{i_1} = v_1, ..., v_{i_k}\}$ be the path (ordered set of vertices) currently

imposed. If the subgraph induced by P contains a Hamiltonian path P_H from v_1 to v_{i_k}, whose cumulative cost is less than that of P, then clearly the current node can be fathomed since none of its descendants could produce the optimal solution. We use a heuristic approach for determining P_H. The method consists of applying to P the 3-opt procedure described in Section 4.2 if $|P| < 8$, and the faster nearest neighbour algorithm otherwise (see, e.g., Rosenkrantz, Stearns and Lewis, 1977). Since the dominance criterion is checked at all nodes, only exchanges involving the last arc of P, $(v_{i_{k-1}}, v_{i_k})$, need be considered in the 3-opt search.

When the node is not fathomed we compute a lower bound $L = \min\{\ell_j : v_j \epsilon V \backslash P\}$, where ℓ_j is a lower bound on the solution we can obtain by extending the current path P through arc (v_{i_k}, v_j). Since ℓ_j is an estimate of the son node bound, when L is less than the value of the incumbent solution, the search is continued by generating the node corresponding to the minimum ℓ_j.

Values ℓ_j are computed through algorithm LB described in Section 3.2. (When the cost matrix (c_{ij}) is symmetrical, the lower bound of Section 3.1, coupled with subgradient optimization, could be used; however, our computational experience indicates that this lower bound, although faster, is generally outperformed by LB.) In our implementation, the multipliers α_k of all son nodes are initially set to the values of the final multipliers of the father node (to zero for the root node). The corresponding ℓ_j values are then computed, the minimum ℓ_j is determined and its associated v_1-arborescence is used for the subgradient updating of the multipliers α_k of all son nodes. Since we use a single multiplier vector $(\alpha_k : v_k \epsilon V)$ for all son nodes,

a parametric computation of the ℓ_j values is possible, with considerable savings in the required computing time.

The number s of iterations was determined after extensive experimentation, as the following function of the branch-decision level $k = |P|$:

$$s(k) = \begin{cases} 2n & if \quad k = 1; \\ \lfloor n/5 \rfloor & if \quad 2 \leq k \leq 5; \\ 1 & if \quad k > 5, \end{cases} \qquad for\ n \leq 25,$$

$$s(k) = \begin{cases} 10n & if\ k = 1; \\ \lceil n/\left(2^{(k-1)/5}\right) \rceil & if\ k > 1, \end{cases} \qquad for\ n > 25.$$

If after $s(k)$ subgradient iterations the gap between the current optimal solution value and the best lower bound encountered is at least halved, $s(k)$ additional subgradient iterations are performed. In any case no more than 1000 subgradient iterations are allowed. For the step-size t along the subgradient direction, we use the classical Held-Karp formula (1971).

At the root node of the branch-decision tree, a greedy algorithm coupled with a 3-opt exchange procedure, is applied to initialize the incumbent solution.

4.2 A Heuristic Algorithm Based on 3-Opt Exchanges

We have developed a two-phase heuristic for the DMP. In the first phase a Hamiltonian circuit is built, starting from v_1, using a nearest neighbour criterion. This choice is particularly suited to our problem since in the objective function, the first arcs on the tour receive a larger weight. In the second phase, the following adaptation of Lin's 3-opt interchange procedure (1965) was implemented.

Recall that this procedure improves a Hamiltonian path or a (directed) tour by repeatedly removing 3 arcs from it and reconnecting the remaining segments in all possible ways. Whenever a reconnection yields an improvement, it is executed. The procedure ends when no such improvements are possible.

For the TSP, the change in the objective function corresponding to a reconnection is easily computed in constant time. This is not the case for the DMP, since any exchange cumulatively affects the cost contribution of all arcs located between the first and last arcs removed. We now show how a simple auxiliary data structure associated with the current tour allows such a computation to be performed in constant time.

Let $(v_{i_1} = v_1, v_{i_2}, ..., v_{i_n})$ be the current tour, and $v_{i_{n+1}} := v_1$. Define

$$C_k := \sum_{t=2}^{k} (n-t+2)\, c_{i_{t-1},i_t} \quad \text{and} \quad L_k := \sum_{t=2}^{k} c_{i_{t-1},i_t}$$

as the cumulative and non-cumulative cost, respectively, of the path from v_1 to v_{i_k} $(k = 2, ..., n+1)$. Then $C_{i_h} - C_{i_k}$ (resp., $L_{i_h} - L_{i_k}$), $k \leq h$, gives the cumulative (resp., non-cumulative) cost of the path from v_{i_k} to v_{i_h}. Note that "moving" this path one position forward (resp. backward) in the tour sequence decreases (resp. increases) its cumulative cost by the quantity $L_{i_h} - L_{i_k}$. Based on this observation, we can compute in constant time the cumulative cost of the alternative tour obtained from the current one by replacing its three arcs $(v_{i_k}, v_{i_{k+1}})$, $(v_{i_h}, v_{i_{h+1}})$ and $(v_{i_q}, v_{i_{q+1}})$, where $1 \leq k < h < q \leq n+1$, by the arcs $(v_{i_k}, v_{i_{h+1}})$, $(v_{i_q}, v_{i_{k+1}})$ and $(v_{i_h}, v_{i_{q+1}})$, see Figure 3. To this end we first define $\alpha := k+1$ and $\beta := \alpha + q - h$ as the new positions of vertices $v_{i_{k+1}}$ and $v_{i_{h+1}}$, respec-

tively, after the exchange. Note that the path from $v_{i_{h+1}}$ to v_{i_q} (resp., from $v_{i_{k+1}}$ to v_{i_h}) is moved $h+1-\alpha$ positions backward (resp. $\beta-k-1$ positions forward) in the tour sequence. Hence the cumulative cost of the tour after the 3-opt exchange is:

$$C_{i_h} + (n-\alpha+2)\, c_{i_h,i_{h+1}} + \left[(C_{i_q} - C_{i_{h+1}}) + (h+1-\alpha)(L_{i_q} - L_{i_{h+1}})\right]$$
$$+(n-\beta+2)\, c_{i_q,i_{k+1}} + \left[(C_{i_h} - C_{i_{k+1}}) - (\beta-k-1)(L_{i_h} - L_{i_{k+1}})\right]$$
$$+(n-q+1)\, c_{i_k,i_{q+1}} + \left[C_{n+1} - C_{i_{q+1}}\right].$$

In addition to the 3-opt exchanges, we try all those involving the removal from the tour of 2 or 3 arcs, and a reconnection in which the vertices of the resulting segments are possibly visited in the reverse sequence. These exchanges are particularly effective when symmetric cost matrices are considered. Again, the cumulative cost of the alternative tour can be computed in constant time, by using a technique (similar to that described above) based on the preliminary computation of the "backward" counterparts of C_k and L_k, namely

$$\widetilde{C}_k := \sum_{t=k}^{n} t\, c_{i_{t+1},i_t} \quad \text{and} \quad \widetilde{L}_k := \sum_{t=k}^{n} c_{i_{t+1},i_t}$$

for $k = n,\ n-1, \ldots, 1$.

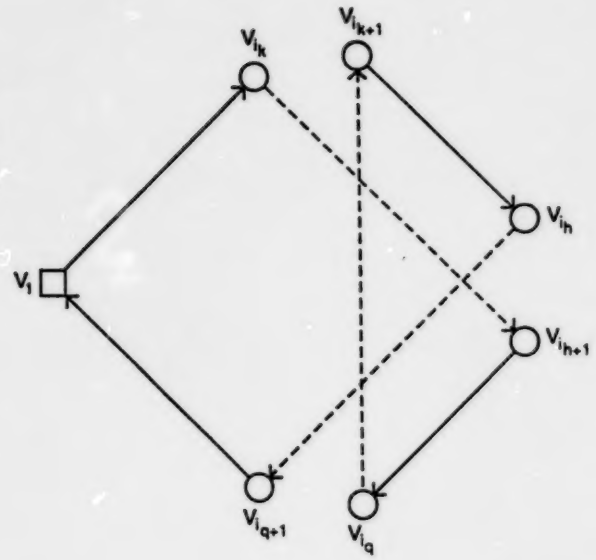

Figure 3. The 3-opt exchange

5. Computational Experiments

The branch-and-bound algorithm was implemented in Fortran and tested on an IBM 6000/320 workstation, on different sets of randomly generated problems.

Three classes of problems were considered. Euclidean symmetric instances (Table 1) were obtained by uniformly randomly generating points coordinates in the 100×100 square and by taking the Euclidean distances (rounded to integers) as costs. Asymmetric instances (Table 2) were obtained by randomly generating integer costs in the range [1,1000] and triangularizing the resulting matrix through shortest paths. Symmetric instances (Table 3) were obtained in the same way, but imposing $c_{ij} = c_{ji}$ for all i, j before triangularization.

For each class and value of n, ten instances were generated. The entries in the tables give the average running time (in CPU seconds), the number of branch-

decision nodes, the percentage ratio lower bound at the root node over optimum, and the percentage ratio heuristic value over the optimum. For the cases where not all instances were solved within the imposed time limit of 1800 seconds, the averages are computed over the solved intances (whose number is reported in brackets).

The tables show that the branch-and-bound algorithm can solve problems up to 60 vertices, and that the more realistic instances of Table 1 are easier to solve. Lucena (1990) (Table III) gives computational results, obtained on a Data General MC1500 computer, for problems generated in the same way as those of Table 1, with n ranging from 15 to 30. Although a direct comparison is not possible, it can be seen that the Lucena lower bound (based on dynamic programming) is tighter. However, the lower bound proposed in this paper can be computed much more efficiently. For example, for $n = 30$, the Lucena algorithm generates an average of 496 nodes and requires 3440 seconds on an MC1500. Our algorithm generates 5352 nodes, but can be executed in only 17.6 seconds on an IBM 6000/320 workstation. The net result is that our algorithm can solve much larger problem sizes than any competing algorithm.

Table 1. Euclidean symmetric instances. IBM 6000/320 computer.
Average values over 10 instances.

n	time	nodes	% LB/optimum	% UB/optimum
10	0.1	53	84.7	101.5
15	0.2	179	91.4	102.8
20	0.4	294	91.3	101.7
25	2.1	1106	91.5	101.2
30	17.6	5352	92.6	103.6
35	36.5	10395	93.1	106.0
40	192.3	38760	93.5	106.6
45	185.3	37940	94.2	104.9
50	450.7	110247	93.9	107.6
55	1309.3(9)	220124	94.7	105.0
60	486.9(5)	66196	94.9	105.3

Table 2. Asymmetric instances. IBM 6000/320 computer.
Average values over 10 instances.

n	time	nodes	% LB/optimum	% UB/optimum
10	0.1	83	74.4	109.4
15	0.7	585	77.1	111.1
20	9.0	6748	75.3	108.4
25	72.8	47753	78.1	108.1
30	44.3	13441	77.2	107.8
35	167.8	40927	73.0	113.2
40	186.1	44175	77.1	115.4
45	740.6(9)	165894	76.5	113.6
50	420.0(5)	100800	77.2	111.7
55	908.6(5)	191714	77.9	114.0

Table 3. Symmetric instances. IBM 6000/320 computer.
Average values over 10 instances.

n	time	nodes	% LB/optimum	% UB/optimum
10	0.1	127	86.5	100.6
15	0.7	689	85.4	100.7
20	2.8	2224	88.3	100.5
25	21.1	13597	85.9	103.2
30	259.1	71476	85.8	102.9
35	461.1	111746	88.5	104.4
40	893.4(6)	213117	86.6	102.5
45	612.0(4)	135556	87.1	104.8

6. Conclusion

The Delivery Man Problem considered in this paper has several important applications in vehicle routing and machine scheduling. It is NP-hard and, in practice, much more difficult to solve than a TSP. We have developed new formulations and bounding schemes for this problem. Their combination within an enumerative algorithm permits the exact solution of problems involving up to 60 vertices, by far the largest size ever attained. In addition, this paper contains two new basic results on the cumulative matroid problem. The first one states that a set optimizing the non-cumulative problem is also optimal for the cumulative case. The second property is that the cumulative matroid problem can be formulated as a linear program. This extends a result by Edmonds for the non-cumulative case.

Appendix

Refer to Section 2 for notation and discussion on the properties of the optimal solution to a cumulative problem. We will show how the cumulative optimization problem on a matroid can be formulated as a linear program. We begin by introducing an integer linear programming model, valid for the general (non-matroidal) case.

For each $H \subseteq E$, define $r(H) := \max\{|I| : I \subseteq H \text{ and } I \epsilon \mathcal{F}\}$. For $k = 1, ..., n$, let $x_j^k = 1$ if item $j \epsilon E$ is among the k smallest cost items in the optimal set I (using any deterministic rule to break ties); $x_j^k = 0$ otherwise. In other words, $x_j^k = 1$ if and only if $x_j \epsilon \{a_1, ..., a_k\}$ in the optimal permutation $(x_i : i \epsilon E)$ of $(a_1, ..., a_n, 0, ..., 0)$; hence $x_j = \sum_{k=1}^{n} \delta_k x_j^k$, where $\delta_k := a_k - a_{k+1}$ for $k = 1, ..., n$ and $a_{n+1} := 0$. Note that the quantity $w_k(I)$, accounting for the sum of the k smallest items in I (see Section 2), can now be expressed as $w_k(I) = \sum_{j \epsilon E} c_j x_j^k$. The cumulative problem is to

$$\text{minimize} \quad \sum_{k=1}^{n} \delta_k \sum_{j \epsilon E} c_j x_j^k, \tag{A.1}$$

subject to

$$\sum_{j \epsilon E} x_j^k = k \qquad (k = 1, ..., n) \tag{A.2}$$

$$x_j^k \leq x_j^{k+1} \qquad (k = 1, ..., n-1;\ j \epsilon E) \tag{A.3}$$

$$\sum_{j \epsilon H} x_j^n \leq r(H) \qquad (H \subseteq E) \tag{A.4}$$

$$x_j^k \epsilon \{0,1\} \qquad (k = 1,...,n;\ j \epsilon E). \tag{A.5}$$

Let $I^k := \left\{ j \epsilon E : x_j^k = 1 \right\}$ $(k = 1,...,n)$. Then (A.2) stipulate that $|I^k| = k$; (A.3) impose that $I^k \subseteq I^{k+1}$ and (A.4) that $I := I^n$ is a feasible set. When $\delta_1 = ... = \delta_{n-1} = 0$ and $\delta_n = 1$, (i.e., $a_1 = ... = a_n = 1$), model (A.1) to (A.5) leads to a well-known version of the non-cumulative problem, in which only variables $x_j^n (j \epsilon E)$ are used.

A surrogate relaxation of problem (A.1)-(A.5) is obtained by replacing (A.3) and (A.4) with

$$\sum_{j \epsilon H} x_j^k \leq r(H) \quad (k = 1,...,n;\ H \subseteq E), \tag{A.6}$$

(note that constraints (A.4) are identical to constraints (A.6) with $k = n$). In this relaxation the cumulative problem splits into n independent non-cumulative subproblems defined on the same subset system.

It is well known that when the subset system $S = (E, \mathcal{F})$ is a matroid, the associated non-cumulative problem can be formulated as the following linear program (Edmonds 1970, 1971):

$$\text{minimize} \qquad \sum_{j \epsilon E} c_j x_j$$

subject to

$$\sum_{j \epsilon E} x_j = n$$

$$\sum_{j \epsilon H} x_j \leq r(H) \qquad (H \subseteq E)$$

$$x_j \geq 0 \qquad (j \epsilon E),$$

always having an optimal integer solution vector x. A similar result holds for the cumulative case, as proven below.

Consider the linear program (LP) defined by (A.1), (A.2), (A.6), and

$$x_j^k \geq 0 \qquad (k = 1, ..., n;\ j \epsilon E). \tag{A.7}$$

We claim that this linear program correctly defines the cumulative problem on a matroid, i.e., that constraints (A.3) and (A.5) are automatically satisfied by an optimal solution to (LP). Indeed, let (LP) be solved by applying the (classical) greedy algorithm to the n independent matroid subproblems associated with vectors x^k, $k = 1, ..., n$ (using the same tie breaking rule at each application): then clearly (A.5) holds. In addition, the greedy nature of the algorithm ensures that constraints (A.3) are satisfied. Hence, the conclusion follows. □

ACKNOWLEDGEMENTS

This work was partially supported by M.U.R.S.T. and C.N.R. (Italy), and by the Canadian Natural Sciences and Engineering Research Council (grant OGP0039682). Thanks are also due to François Dehaut for his help with programming.

REFERENCES

Afrati, F., S. Cosmadakis, C.H. Papadimitriou, G. Papageorgiou and N. Papakonstantinou. 1986. The Complexity of the Traveling Repairman Problem. *Theoretical Informatics and Applications* 20, 79–87.

Balakrishnan, A., T.L. Magnanti and R.T. Wong. 1989. A Dual-Ascent Procedure for Large-Scale Uncapacitated Network Design. *Operations Research* 37, 716–740.

Bianco, L., A. Mingozzi and S. Ricciardelli. 1988. Minimizing Flowtime with Sequence Dependent Processing Times. Working paper R.224, Istituto di Analisi dei Sistemi ed Informatica, Consiglio Nazionale delle Ricerche, Italy.

Desrochers, M. and G. Laporte. 1991. Improvements and Extensions to the Miller-Tucker-Zemlin Subtour Elimination Constraints. *Operations Research Letters*. Forthcoming.

Edmonds, J. 1967. Optimum Branchings. *Journal of Research of the National Bureau of Standards* 71B, 233–240.

Edmonds, J. 1970. Submodular Functions, Matroids and Certain Polyhedra. In R. Guy *et al.*, eds., *Combinatorial Structures and Their Applications*, Gordon and Breach, 69–87.

Edmonds, J. 1971. Matroids and the Greedy Algorithm. *Mathematical Programming* 1, 127–136.

Fischetti, M. and P. Toth. 1988. A New Dominance Procedure for Combinatorial Optimization Problems. *Operations Research Letters* 7, 181–187.

Fischetti, M. and P. Toth. 1989. An Efficient Algorithm for the Min-Sum Arborescence Problem. *Strumenti per lo Sviluppo della Innovazione Tecnologia nelle Imprese e nel Governo del Territorio*. Atti giornate AIRO, Udine, Italy, 197–215.

Garey, M.R. and D.S. Johnson. 1979. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman, San Francisco.

Grötschel, M. and M.W. Padberg. 1985. Polyhedral Theory. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys, eds., *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization*. Wiley, Chichester, 251–305.

Guignard, M. and S. Kim. 1987. Lagrangean Decomposition for Integer Programming: Theory and Applications. Technical Report 93, Department of Statistics, The Wharton School, University of Pennsylvania.

Held, M. and R.M. Karp. 1971. The Traveling-Salesman Problem and Minimum Spanning Trees: Part II. *Mathematical Programming* 1, 6–25.

Jørnsten, K. and M. Nasberg. 1986. A New Lagrangean Relaxation Approach to the Generalized Assignment Problem. *European Journal of Operational Research* 27, 313–333.

Lawler, E.L. 1976. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, New York.

Lin, S. 1965. Computer Solutions of the Traveling Salesman Problem. *Bell System Technical Journal* 44, 2245–2269.

Lucena, A. 1990. Time-Dependent Traveling Salesman Problem – The Deliveryman Case. *Networks* 20, 753–763.

Martello, S. and P. Toth. 1990. *Knapsack Problems: Algorithms and Computer Implementations*, Wiley, Chichester.

Miller, C.E., A.W. Tucker and R.A. Zemlin. 1960. Integer Programming Formulations and Traveling Salesman Problems. *Journal of the Association for Computing Machinery* 7, 326–329.

Minieka, E. 1989. The Delivery Man Problem on a Tree Network. In F.V. Louveaux, M. Labbé, J.-F. Thisse, eds., *Facility Location Analysis: Theory and Applications, Annals of Operations Research* 18, 261–266.

Papadimitriou, C.H. and K. Steiglitz. 1982. *Combinatorial Optimization. Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ.

Picard, J.-C. and M. Queyranne. 1978. The Time-Dependent Traveling Salesman Problem and its Applications to the Tardiness Problem in One-Machine Scheduling. *Operations Research* 26, 86–110.

Ribeiro, C. and M. Minoux. 1985. Solving Hard Constrained Shortest Path Problems by Lagrangean Relaxation and Branch and Bound Algorithms. *Methods of Operations Research* 53, Anton Hain, 303–316.

Rinnooy Kan, A.H.G. 1976. *Machine Scheduling Problems*. Martinus Nijhoff. The Hague.

Rosenkrantz, D.J., R.E. Stearns and P.M. Lewis II. 1977. An Analysis of Several Heuristics for the Traveling Salesman Problem. *SIAM Journal on Computing* 6, 563–581.

Sahni, S. and T. Gonzalez. 1976. P-Complete Approximation Problems. *Journal of the Association for Computing Machinery* 23, 555–565.

Simchi-Levi, D. and O. Berman. 1991. Minimizing the Total Flow Time of n Jobs on a Network. *IIE Transactions*. Forthcoming.

Tsitsiklis, J. 1990. Special Cases of Traveling Salesman and Repairman Problems with Time Windows. Working Paper. Laboratory for Information and Decision Systems and Operations Research Center, Massachusetts Institute of Technology.

Wong, R.T. 1984. Dual Ascent Approach for Steiner Tree Problems on a Directed Graph. *Mathematical Programming* 28, 271–287.